



MASTER IN HIGH PERFORMANCE COMPUTING

Increasing speedup of naval hull workflow optimization using Reduced Order Methods

Supervisor(s):

Dott. Luca HELTAI,

Ing. Giampiero LAVINI

Candidate:

Nicola DEMO

2nd EDITION

2015–2016

Contents

1	Introduction	1
2	Reduced Order Methods	3
2.1	Algebraic Reduction	4
2.2	Offline-Online procedure	4
2.3	Reduced basis space generation	5
2.4	A posteriori error estimation	5
3	Shape Design Optimization System	7
3.1	Shape Design Parametrization	8
3.1.1	Free-form Deformation	9
3.1.2	PyGeM	10
3.1.3	Mesh versus Geometry Deformation	10
3.2	Computational Fluid Dynamics	12
3.2.1	Turbulent flow	13
3.2.2	Finite Volume Method	14
3.2.3	OpenFOAM	14
3.3	Optimization	14
3.3.1	Algorithms	14
3.3.2	Dakota	15
4	Reduced Order Method Injection	17
4.1	EZyRB	17
4.2	Mapped Solution	18
4.2.1	Mapping algorithm	18
4.2.2	Error Handling	20
4.3	Reduced Basis Space Improvement	22
4.3.1	Global	22
4.3.2	Local	23
5	Results	25
5.1	Configuration	25
5.2	Parametrization	25
5.3	Mapped solution	26
5.4	Solver	28
5.4.1	Reduced Basis Optimization	28
5.4.2	Reduced Basis Error	32
5.4.3	Reduced Basis Speedup	33
6	Conclusion	35

Chapter 1

Introduction

The bulb of a ship is a key element in lowering its resistance to motion. The resistance of a hull is divided into two components. The first component, having a viscous origin, is due to the friction of the water on the hull, and it acts tangentially to the surface. The second component is defined pressure component and it acts normally to the surface of the hull. This pressure component is higher when the hull forms are filled and it is mainly due to the wave formation created by the vessel in its advancement.

The bow bulb aims essentially to create a wave in phase opposition with respect to that of the ship. This bulb, extending the beyond the bow and under the water surface, has to be built in order to produce a secondary wave that must have the minimum corresponding to the maximum of the hull's wave; moreover the heights of these waves have to have similar values. The result of the combination of the two waves is a strongly pulled down wave, that yields a significant reduction of the resistance; optimal bulbs can reduce the ship resistance of well above 20%.

Because of the depending of the hull shape, the bulb optimization is a challenging task in naval engineering: for each different hull, evaluation of a large number of solutions is needed. Manually, this would require a too expensive effort; fortunately, the geometry of a bulb can be easily parametrized through the definition of some features (e.g., length, width, immersion). That means the study of the bulb becomes an optimization procedure of parametrized Partial Differential Equations (PDEs). Optimization problem is generally understood as the problem of finding the parameter combination which minimizes a given object function. This implies an iterative process where several PDEs are solved with ad high fidelity solver (i.e., finite elements, finite volumes, Galerkin discontinuous methods) until the optimal combination of parameters is obtained. Two problems occur: *(i)* dependently on the problem to solve and on the number of parameters, several PDEs need to be solved and *(ii)* at each optimization iteration, the PDE is solved from scratch, even when the solution changes smoothly with the parameters. This implies the decreasing of the overall computational efficiency.

Model order reduction techniques, such as reduced basis methods [1, 2], are a nice solution strategy to achieve this goal: they allow possible a strong reduction of computational cost required when solving parametric PDE prob-

lems, owing to a crucial decomposition of the computational procedures. In an offline pre-processing stage, a suitable basis is stored by solving the original problem for a set of parameter values, properly selected in an automatic and optimal way. During an online stage, for each new parameter value the solution is found as a combination of the previously computed basis functions, by means of a Galerkin projection [1]. This problem has a very small size (related with the number of the selected bases, which are typically very few). The resulting procedure is not only rapid and efficient but also accurate and reliable, thanks to residual-based a posteriori error estimators.

During the work presented in this thesis, we apply Reduced Order Methods (ROMs) to a naval industrial problem, coupling a suitable libraries to a finite volumes solver in order to increase the speedup of shape design optimization workflow. The purpose of this work is to analyze the result of the optimization workflow in complex system (viscous fluid, boundary layer), before and after ROM injection, from the computational time and the numerical accuracy point of view. The thesis is organized as follows: we present a deeper discussion about Reduced Basis (RB) methods in Chapter 2, then in the Chapter 3 we provide an overall idea of a standard optimization workflow. In Chapter 4 we focus about the re-organization of this system after ROM injection and so in Chapter 5 we present some results collected, focusing on the speedup of the optimization procedure and the on accuracy of the ROM-injected optimization workflow. Finally, in Chapter 6 we summarize give some conclusions and a brief outlook of possible development.

Chapter 2

Reduced Order Methods

Reduced Order Method (ROM) allows possible an efficient computational reduction by computing a low-dimensional approximation of a parameter dependent high-fidelity solution (here indicated with $\boldsymbol{\mu}$) in an inexpensive way. Exploiting Reduced Basis (RB) methods, we express the solution of a Partial Differential Equation (PDE), for any new value $\mu \in \mathcal{D} \subset \mathbb{R}^G$, as a linear combination of suitable basis functions. The space of the parameters where these basis functions belong to is originated by a linear combination of high-fidelity solutions, computed for suitable parameter values.

We formally define the high-fidelity system of a generic PDE in the form:

$$a(y^{\mathcal{N}}(\boldsymbol{\mu}), w; \boldsymbol{\mu}) = F(w, \boldsymbol{\mu}), \quad \forall w \in \mathbb{X}^{\mathcal{N}} \quad (2.1)$$

where $y^{\mathcal{N}}(\boldsymbol{\mu}) \in \mathbb{X}^{\mathcal{N}}$ is the solution, and \mathcal{N} is the number of degrees of freedom in discretized approximation.

The RB method allows efficient computation of the reduced solution $y_N^{\mathcal{N}}(\boldsymbol{\mu})$ as approximation of $y^{\mathcal{N}}(\boldsymbol{\mu})$ by using a Galerkin projection on a reduced subspace made of well-chosen high-fidelity solutions.

Defining

$$S_N = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_N\} \quad (2.2)$$

as the set of suitable parameter values of order N , the reduced space is

$$\mathbb{X}_N^{\mathcal{N}} = \text{span}\{\xi_n \mid \xi_n \equiv y^{\mathcal{N}}(\boldsymbol{\mu}_n)\}, \quad \forall \boldsymbol{\mu}_n \in S_N. \quad (2.3)$$

Hence, the reduced formulation of problem (2.1) becomes:

$$a(y_N^{\mathcal{N}}(\boldsymbol{\mu}), w; \boldsymbol{\mu}) = F(w, \boldsymbol{\mu}), \quad \forall w \in \mathbb{X}_N^{\mathcal{N}} \quad (2.4)$$

where $y_N^{\mathcal{N}}(\boldsymbol{\mu})$ refers to the reduced solution.

While the \mathcal{N} degrees of freedom used to discretize the high-fidelity problem is usually a very big number, the solutions to build the reduced space are typically very few. Thanks to the reduced dimension N of the linear system obtained from RB method, we can provide a reliable approximation of the solution in a rapid way, saving computational time, at the cost of solving a few high-fidelity problems.

2.1 Algebraic Reduction

Let's recall (2.3) the definition of reduced space $\mathbb{X}_N^{\mathcal{N}}$ given by the reduced basis $\{\xi_n\}_{n=1}^N$; we define the matrix $\mathbf{B} \in \mathbb{R}^{N \times N}$ such that:

$$\xi_n = \sum_{i=1}^N \mathbf{B}_{i,n} \varphi_i. \quad (2.5)$$

This means that the n -th reduced basis ξ_n is a linear combination of the high-fidelity basis functions (of the discretized problem) $\{\varphi_i\}_{i=1}^N$, where the coefficients are represented in the n -th column of \mathbf{B} .

Then, we define the reduced basis solution matrix $\mathbf{A}_N(\boldsymbol{\mu}) \in \mathbb{R}^{N \times N}$ and the reduced right hand side $f_N(\boldsymbol{\mu}) \in \mathbb{R}^N$ as:

$$(\mathbf{A}_N(\boldsymbol{\mu}))_{m,n} := a(\xi_n, \xi_m; \boldsymbol{\mu}), \quad (f_N(\boldsymbol{\mu}))_m := F(\xi_m, \boldsymbol{\mu}), \quad 1 \leq n, m \leq N. \quad (2.6)$$

Taking the equation (2.5), we obtained:

$$\mathbf{A}_N(\boldsymbol{\mu}) := \mathbf{B}^T \mathbf{A}(\boldsymbol{\mu}) \mathbf{B} \quad (2.7)$$

$$f_N(\boldsymbol{\mu}) := \mathbf{B}^T f(\boldsymbol{\mu}) \quad (2.8)$$

where \mathbf{A} is the solution matrix and f is the right hand side of the discretized problem (2.1). Hence, the reduced basis approximation $y_N^{\mathcal{N}}(\boldsymbol{\mu}) = \sum_{n=1}^N (y_N^{\mathcal{N}})_n \xi_n$, where $\{(y_N^{\mathcal{N}})_n\}_{n=1}^N$ are the coefficients of the vector $y_N^{\mathcal{N}}(\boldsymbol{\mu})$, is the solution of the linear system

$$\mathbf{A}_N(\boldsymbol{\mu}) y_N^{\mathcal{N}} \boldsymbol{\mu} = f_N(\boldsymbol{\mu}) \quad (2.9)$$

2.2 Offline-Online procedure

As discussed in the previous sections, a reduced basis solution combines the already computed high-fidelity solutions, below called *snapshots*. The solution of several PDEs, is required to create the *database* where the snapshots will be stored. We split the ROM in two different parts:

- **Offline:** the most expansive part, required to generate the reduced basis space: several PDEs are solved using the high-fidelity solver, each with a suitable configuration, and the results are used for space generation.
- **Online:** the inexpensive and rapid part; solution for a query configuration is obtained by projection onto the previously created reduced basis space.

The offline part is executed just once, while the online part can be computed many and many times. After the reduced basis space generation the Galerkin projection requires very little time. The RB method does not provide only an efficient reduction of the computational time, but it allows a immediate response in real-time and multi-query contexts.

Moreover, a further not-negligible gain comes; nowadays, most of the scientific code runs on distributed supercomputers. Thanks to the Offline-Online

subdivision, ROM gains an additional bonus: the Offline part, requiring hours and hours of CPU time, can be executed on High Performance Computing (HPC) clusters, while the Online part, typically computed by final users, can be rapidly computed on an ordinary personal computer. Cutting-edge (and expensive) supercomputer is exploited by running the high computational part, and let the query approximation be *supercomputer-free*.

2.3 Reduced basis space generation

There are several strategies for generating the reduced basis space; in our work, we will focus (and use) only the Proper Orthogonal Decomposition (POD).

POD is an explore-and-compress strategy in which the numerical solutions are computed for some sample points belonging to parametric space and, following compression, only the solutions more energetic are compressed and store into modes. In this way, the dimensionality of a system is reduced by transforming the original variables into a new set of uncorrelated variables, called POD modes.

Consider a set of sample points $S_{Ns} = \{\boldsymbol{\mu}_n\}_{n=1}^{Ns}$, of dimension $|S_{Ns}| = Ns$; we define $\mathcal{S} \in \mathbb{R}^{N \times Ns}$ as the matrix containing the snapshots computed for the sample points, by column:

$$\mathcal{S} = [y^N(\boldsymbol{\mu}_1), \dots, y^N(\boldsymbol{\mu}_{Ns})] \quad \text{with} \quad y^N(\boldsymbol{\mu}_1), \dots, y^N(\boldsymbol{\mu}_{Ns}) \in \mathbb{X}^N \quad (2.10)$$

Then, the matrix \mathcal{S} is decomposed using Singular Value Decomposition (SVD), such that

$$\mathcal{S} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V} \quad (2.11)$$

where

- $\mathbf{U} \in \mathbb{R}^{N \times N}$ is unitary and orthogonal matrix;
- $\mathbf{V} \in \mathbb{R}^{Ns \times Ns}$ is unitary and orthogonal matrix;
- $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times Ns}$ is diagonal matrix.

The elements on the diagonal of $\boldsymbol{\Sigma}$ are ascending ordered until the r -th element: $r \leq Ns$ is the rank of \mathcal{S} . For any $N \leq r$, the POD basis of order N is defined as the first N left singular vectors of the matrix \mathbf{U} :

$$\mathbf{U} = [\xi_1, \xi_2, \dots, \xi_N] \quad (2.12)$$

$$\mathbf{B} = [\xi_1, \xi_2, \dots, \xi_N] \in \mathbb{R}^{N \times N} \quad (2.13)$$

2.4 A posteriori error estimation

We define the reduced basis approximation error

$$e(\boldsymbol{\mu}) = y^N(\boldsymbol{\mu}) - y_N^N(\boldsymbol{\mu}), \quad e(\boldsymbol{\mu}) \in \mathbb{X}^N. \quad (2.14)$$

Without going into details of the rigorous error boundary formulation, exhaustively treated in [1], we introduce an efficient and powerful way to *estimate* the reduced basis error.

We remark that given the set $S_N = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_N\}$ of suitable points belonging to the parametric space, the reduced basis space is generated as a linear combination of high-fidelity solutions computed at these points.

$$\mathbb{X}_N^{\mathcal{N}} = \text{span}\{y^{\mathcal{N}}(\boldsymbol{\mu}_i)\}, \quad \forall \boldsymbol{\mu}_i \in S_N \quad (2.15)$$

The key idea to the efficient error estimation is to reuse the already computed numerical solutions to generate several reduced basis spaces, each of them made up by linear combination of the high-fidelity solutions except one. Formally:

$$(\mathbb{X}_N^{\mathcal{N}})_j = \text{span}\{y^{\mathcal{N}}(\boldsymbol{\mu}_i)\}, \quad \boldsymbol{\mu}_i \in S_N, i \neq j \quad (2.16)$$

Hence, the error at j -th point of S_N can be easily computed as the difference between the high-fidelity solution in that point and the reduced basis approximation in the proper space $(\mathbb{X}_N^{\mathcal{N}})_j$

$$e(\boldsymbol{\mu}_j) = y^{\mathcal{N}}(\boldsymbol{\mu}_j) - \eta_N^{\mathcal{N}}(\boldsymbol{\mu}_j), \quad \eta_N^{\mathcal{N}}(\boldsymbol{\mu}_j) \in (\mathbb{X}_N^{\mathcal{N}})_j, j = 1, \dots, N_S. \quad (2.17)$$

Finally, the error for each parametric point is computed. The maximum provides us an estimation of the reduced approximation accuracy.

$$\hat{e}(\boldsymbol{\mu}) = \arg \max_{1 \leq j \leq N_S} \{ \| e(\boldsymbol{\mu}_j) \| \}. \quad (2.18)$$

This method does not give us an exact error, but it provides an ideal estimation; an important feature is the fast computational time required: considering the N_S high-fidelity solutions necessary for reduced space generation, this method runs N_S space generations and N_S query approximations. The time required for these operations is negligible compared to the high-fidelity solver time.

Chapter 3

Shape Design Optimization System

In this chapter, we discuss exhaustively about the initial optimization workflow: at this stage, ROM is not yet involved, but we consider extremely relevant to introduce the starting optimization system for two main reasons: (i) to show concretely how the system evolves in order to reduce computational effort and (ii) to introduce the involved components, because we will reuse them in the final system.

The goal of an optimization problem is to minimize a given cost functional $\mathcal{J}(\Omega, y(\Omega))$ by finding the optimal shape of the domain where the PDEs problem is defined. Given a set of possible configuration \mathcal{O}_{ad} , the optimal shape is defined

$$\hat{\Omega}_{\mathcal{O}} = \arg \min_{\Omega_{\mathcal{O}} \in \mathcal{O}_{\text{ad}}} \mathcal{J}(\Omega_{\mathcal{O}}, y(\Omega_{\mathcal{O}})). \quad (3.1)$$

Hence, a shape optimization framework should be composed by three main components:

1. a parametrization tool that allows to obtain the shape $\Omega_{\mathcal{O}}$ by deforming the original shape Ω ;
2. an high fidelity solver to compute the solution $y(\Omega_{\mathcal{O}})$;
3. an optimization algorithm to looking for the best shape in \mathcal{O}_{ad} .

The figure 3.1 represents the algorithm of a generic workflow composed by the above mentioned components. Taking as input the original geometry, the iterative procedure runs, according to the optimization algorithm chosen, several simulations: for each simulation, the geometry is deformed by a suitable parameter and the solver calculates the numerical solution of the current configuration. Finally, The objective function is evaluated on each configuration and it is used to optimal convergence.

The objective function are very relevant: it has to express the important properties of the case analyzed. In our case, we are looking for the optimal shape of the ship for resistance minimization: our aim is the reduction of the forces which act on the shape. So the chosen object function is the following:

$$\mathcal{J}(\Omega_{\mathcal{O}}, y(\Omega_{\mathcal{O}})) = \int_{\Omega_{\mathcal{O}}} \rho n \, dx + \int_{\Omega_{\mathcal{O}}} \tau \, dx \quad (3.2)$$

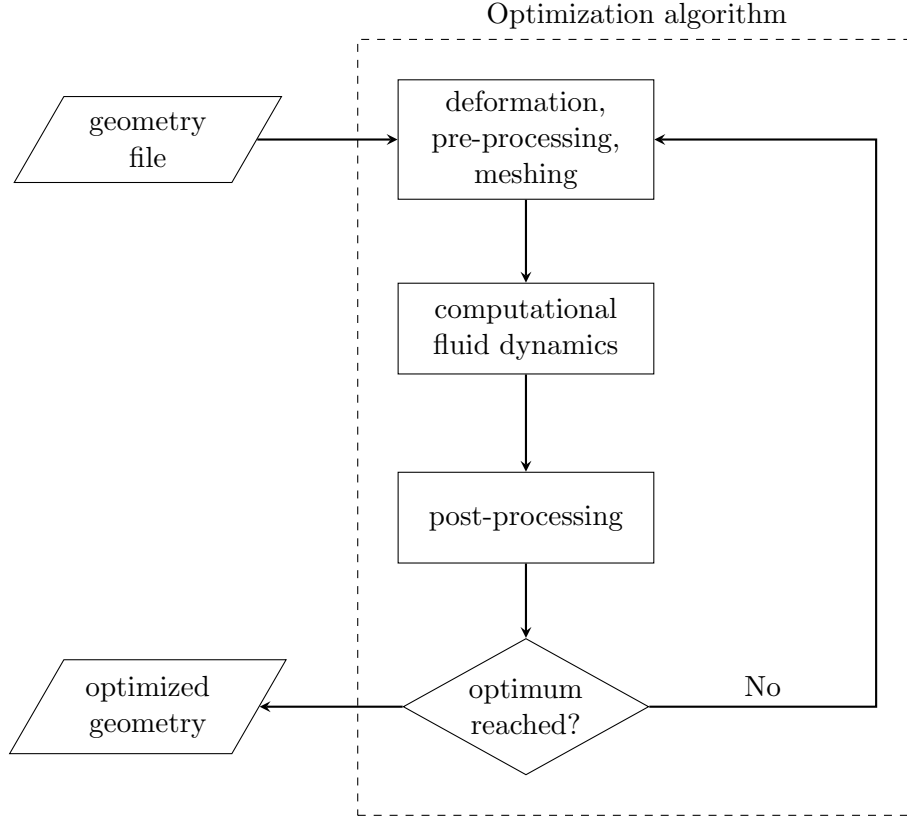


Figure 3.1: Optimization workflow

where ρ is the pressure on ship surface, n refers to the surface normals and τ to the wall shear stress; so, what we want to minimize is the integral of drag and lift forces among the x direction (the direction of the ship motion).

At this stage, the system structure is not really complex: provided you have good tools for each components (domain parametrization, PDEs solver, optimization algorithm), it is possible to build a working workflow just coupling different software. The problems occur when looking at the computational cost of this system: at each iteration, a new PDE is solved from scratch and may require several hours; and the number of iterations (depending from the physical problem, from the number of parameters and from the algorithm) may vary from dozens to thousand. At the moment, we focus on the different components composing the system, going deeper in the idea behind and the implementation; in the next chapter (4) we explain how modify this system to obtain optimal design avoiding months of calculations.

3.1 Shape Design Parametrization

Our work is based on the bulbous bow deformation and optimization. The bulb geometry can be easily parametrized through the definition of the main features; moreover, from the engineering point of view, complex shapes are useless in the optimization study. The only constraints is that the deformation has to preserve the derivatives continuity on the surface.

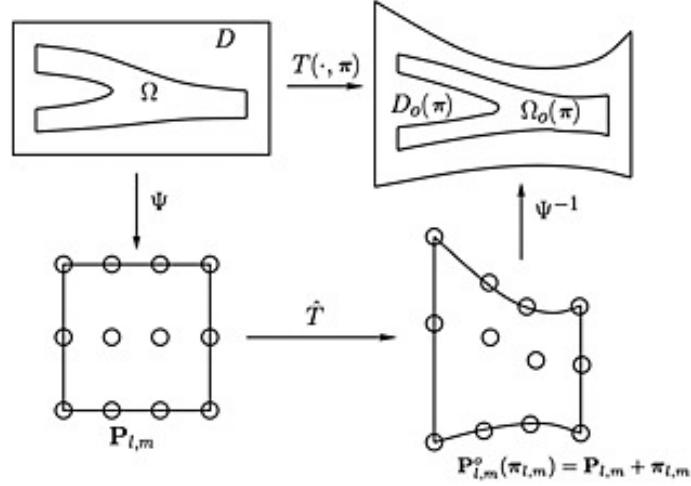


Figure 3.2: Free-form Deformation

3.1.1 Free-form Deformation

Free-form Deformation (FFD) is a modeling technique introduced in the late 80's [6]. Initially it was widely used in computer graphics. Its qualities are the great versatility and the inexpensive computational cost, and it was be exploited in the last years for ROM.

FFD acts on a bivariate Bezier space, built around to the objects to deform. The key idea is to manipulate this space by moving a lattice of points, changing the shape of the objects. The parametric values do not refer to the geometrical properties of the objects, but they represents the directions of the control points displacement. It allows to manage the continuity of the derivatives in the deformed surface by *anchoring* some control points.

Given a fixed rectangular domain $D \subset \mathbb{R}^d$ that contains the domain to deform Ω , we assume the existence of an affine function $\Psi(x)$ that maps D to the reference hypercube $\bar{D} = [0, 1]^d$. Also, we introduce a set of control points called *FFD control points* $\{P_j\}_{j=1}^{N_g}$, where $N_g := \prod_{k=1}^d N_{g,k}$, and $N_{g,k}$ is the number of *FFD control points* in the coordinate direction k (figure 3.2). The deformed position of j -th control point is obtained as $P_j + \mu_j$. Since it's possible for some FFD control points to be fixed or to move only in some prescribed coordinate direction, the parameter vector $\mu \in \mathbb{R}^G$ will contain only non-zero displacement components, and $G \leq dN_g$.

The *Free-form Deformation* map $T(\cdot; \mu) : D \rightarrow \mathbb{R}^d$ is defined as:

$$T(x; \mu) = \Psi^{-1}(\bar{T}(\Psi(x); \mu)) \quad (3.3)$$

where $\bar{T}(\cdot; \mu) : \bar{D} \rightarrow \mathbb{R}^d$ is:

$$\bar{T}(\bar{x}; \mu) = \sum_{j=1}^{N_g} b_j(\bar{x})[P_j + \mu_j] \quad (3.4)$$

and $b_j(\bar{x})$ is the tensor product of one-dimensional Bernstein polynomials

$$b_j(\bar{x}) = b_{j_1}(\bar{x}_1), \quad (3.5)$$

$$b_{j_k}(\bar{x}_k) = \binom{N_k}{j_k} (1 - \bar{x}_{j_k})^{N_{g,k} - j_k} \bar{x}_k^{j_k}. \quad (3.6)$$

Finally, the parametrized domain is obtained applying the *Free-form Deformation* map:

$$\Omega_0(\mu) = T(\Omega; \mu) \quad (3.7)$$

3.1.2 PyGeM

PyGeM [9] is an open source library written in Python that uses Free Form Deformation to parametrize complex geometries. It allows to handle Computer Aided Design files, Mesh files and Output files in their most common formats, such as VTK files or STL files. The library allows to control geometry deformation just by simple text file: dimension and position of domain D and number of FFD control points long each direction, FFD control points movement and so on can be controlled via parameter files. These features make PyGeM specially suited for actual industrial problems.

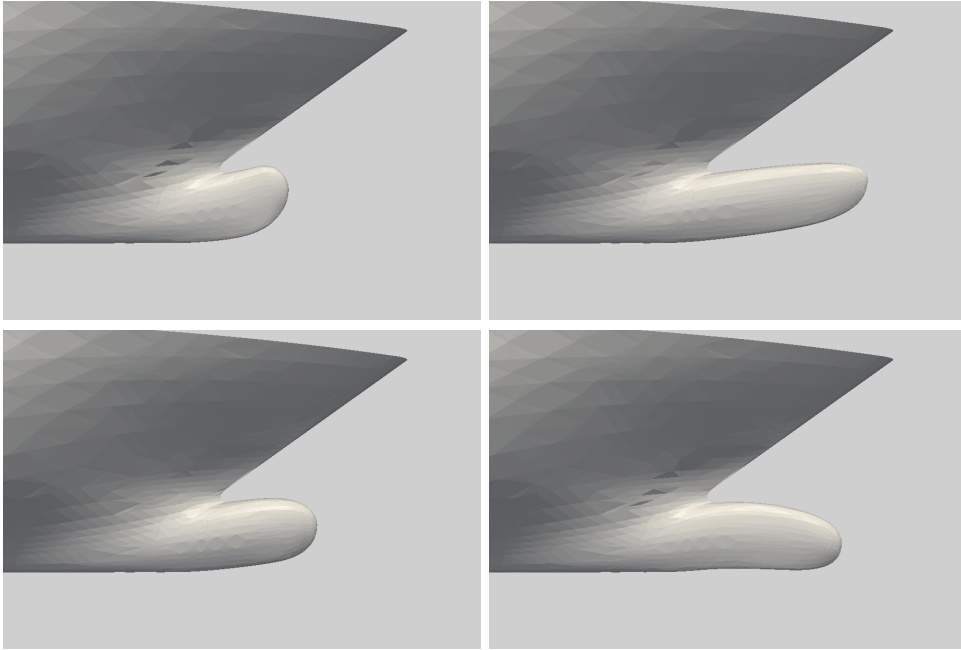


Figure 3.3: Examples of hull deformed using PyGeM

The snippet shown in 3.1 introduce a basic Python script that performs a FFD: points defining the surface are extracted from the input files (here a STL file), and using the configuration read from file they are deformed. Finally, the result points are stored in the output file.

The library is simple to use, it implements all the FFD features: Figure 3.3 shows some examples created using a Python script quite similar to one presented above and a proper file for the parameters configuration.

3.1.3 Mesh versus Geometry Deformation

Free-form Deformation operates on the domain D by FFD control points movement. In this way, every object inside the domain is deformed: this technique is successfully exploited to perform IGA points deformation in isogeometric analysis-based [7] and to perform mesh point deformation [4]. This

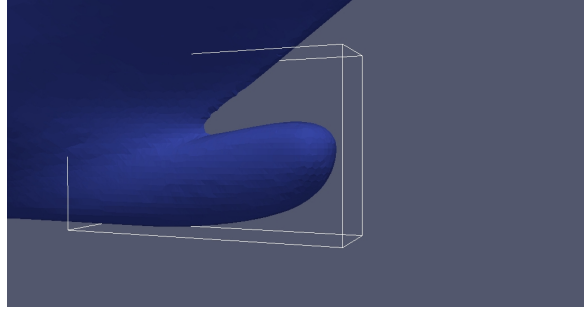
Listing 3.1: Example of *Free-form Deformation* on STL file using PyGeM

```

1  import pygem
2
3  params = pygem.params.FFDParameters()
4  params.read_parameters('parameters.file')
5
6  stl_handler = pg.stlhandler.StlHandler()
7  mesh_points = stl_handler.parse('input_geometry.stl')
8
9  free_form = pg.freeform.FFD(params, mesh_points)
10 free_form.perform()
11 new_mesh_points = free_form.modified_mesh_points
12 stl_handler.write(new_mesh_points, 'output_geometry.stl')

```

implies an extra benefit that we have not talked yet: the possibility to deform the mesh reduces the overall computational cost by avoiding the discretization for each parametrized shape.

Figure 3.4: Representation of deforming domain D

From the computational point of view, the geometric deformation looks completely useless, considering that it is possible to reach the same result in less time by deforming the mesh. However, mesh deformation is not always the best solution for the shape parametrization; we remark that in our work we need to modify the bulbous bow, leaving the remaining surface unchanged. A possible domain D where the FFD acts is shown in Figure 3.4.

Trying to deform only a local region of the mesh, some problems may occur: if we define the FFD domain D to include not the complete mesh (as in example above mentioned) and we compute the FFD, several cells become irregulars (Figure 3.5). At the boundary of the domain, all the cells have been stretched to preserve the continuity of the mesh, while in the middle, the volume (or the area) of the cells is obviously increased or decreased. This can create, using a high-fidelity solver, instability in the solution.

One possibility to avoid this behavior is to include the complete mesh in the FFD domain, and using much more FFD control points to deform just a portion of the mesh. This strategy requires a bigger effort to computing the wanted deformation, and anyway it does not ensure that the deformed mesh will have the same features of the original one.

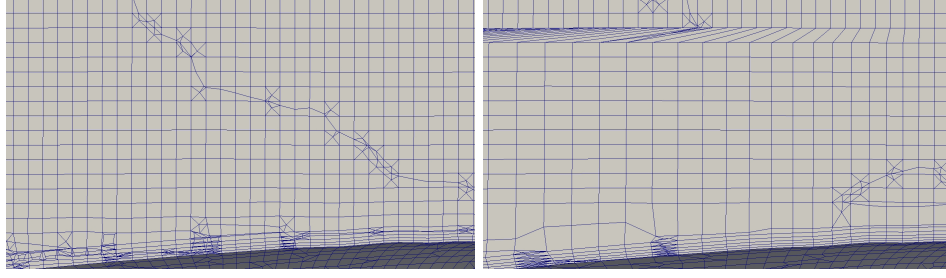


Figure 3.5: Mesh quality comparison between the surface deformation (left) and the mesh deformation (right)

To conclude, the mesh deformation may reduce the mesh quality and compromise the accuracy of the numerical solution; to avoid this, without changing the meshing procedure, the deformation of the original shape is needed: the mesh will be created on the new geometry, preventing any loss of quality, at the expense of increased computational cost. In Figure 3.5, the differences between these two methods are presented: by deforming the mesh, several cells become more bigger than wanted, and at boundary they are stretched; instead, by generating mesh on the deformed geometry (using the same parameter values) the cells look regular and the global quality of the mesh is appreciably better.

3.2 Computational Fluid Dynamics

The goal of a fluid dynamics simulation is to replicate the fluid behavior in order to analyze it. This means solving PDEs that reproduce the fluid motion: Navier-Stokes equations are widely used to model incompressible viscous fluid description.

$$\begin{cases} \frac{\partial \mathbf{U}_i}{\partial x_i} = 0 \\ \frac{\partial \mathbf{U}_i}{\partial t} + \mathbf{U}_j \frac{\partial \mathbf{U}_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 \mathbf{U}_i}{\partial x_j \partial x_j} \end{cases} \quad (3.8)$$

Equation 3.8 shows Navier-Stokes equation for incompressible Newtonian fluid, with \mathbf{U} , p , ρ and ν denote respectively the fluid velocity, the pressure, the fluid density and the kinematic viscosity; analytical solution for this kind of equation is usually impossible to reach, so computational approximation is mandatory.

The purpose of this work is not focusing about the physical equations or analyze the numerical approximation methods; we introduce in this section an overview about the simulation computed during optimization procedure, by recall the demonstration of the Navier-Stokes equation solved and the numerical approximation used to reach the solution.

3.2.1 Turbulent flow

Reynolds-averaged Naviers-Stokes

For high Reynolds Number \mathbf{Re} , the flow enters in turbulent regime and smaller eddies will form. The eddies correspond to small random fluctuations in the variables describing the flow. Hence, in a turbulent flow the velocity field $\mathbf{U}(x, t)$ can be expressed by a mean and a fluctuating part (Reynolds decomposition):

$$\mathbf{U}(x, t) = \overline{\mathbf{U}(x, t)} + \mathbf{U}'(x, t) \quad (3.9)$$

with x and t denote respectively spatial and temporal coordinates. The $\overline{\mathbf{U}}$ stand for the time average of \mathbf{U} , while \mathbf{U}' the fluctuating part. Moreover, the time average of the fluctuating part has to be equal to zero.

$$\overline{\mathbf{U}(x, t)} = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T \mathbf{U}(x, t) dt, \quad (3.10)$$

$$\overline{\mathbf{U}'(x, t)} = 0 \quad (3.11)$$

We insert the Reynolds decomposition idea (3.9) to the original Navier-Stokes equation (3.8) to arrive at the Reynolds-averaged Naviers-Stokes (RANS):

$$\begin{cases} \frac{\partial \overline{\mathbf{U}}_i}{\partial x_i} = 0 \\ \frac{\partial \overline{\mathbf{U}}_i}{\partial t} + \overline{\mathbf{U}}_j \frac{\partial \overline{\mathbf{U}}_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{\mathbf{U}}_i}{\partial x_j \partial x_j} - \frac{\partial \overline{\mathbf{U}}'_i \mathbf{U}'_j}{\partial x_j} \end{cases} \quad (3.12)$$

The important difference between (3.8) and (3.12) is the introduced term $\overline{\mathbf{U}}'_i \mathbf{U}'_j$, known as Reynolds stresses tensor R_{ij} .

Turbulent models

Let recall the definition of Navier-Stokes equations (3.8): the system has four equations (the continuity equation and the three components for the x, y, z direction) and there are four unknowns. Therefore, the system is closed. By introducing the Reynolds stress R_{ij} , new unknowns have been introduced without adding equations: the system is not anymore closed (closure problem).

Hence, to solve RANS equations, R_{ij} has to be approximated as a function of the mean part of equation. The first hypothesis to resolve the unclosure problem was defined by Boussinesq:

$$R_{ij} = \overline{\mathbf{U}}'_i \mathbf{U}'_j = \frac{2}{3} k \delta_{ij} - \nu_T \left(\frac{\partial \overline{\mathbf{U}}_i}{\partial x_j} + \frac{\partial \overline{\mathbf{U}}_j}{\partial x_i} \right) \quad (3.13)$$

Starting from this hypothesis, many models were developed to improve the accuracy of the numerical approximation. In our work, we adopt the popular Menter's Shear Stress Transport (SST) model [8]. This model is a two-equations eddy-viscosity that combines two already existing models, $k - \varepsilon$ turbulence model and $k - \omega$ turbulence model.

3.2.2 Finite Volume Method

The Finite Volume (FV) method is a technique for numerical approximation of PDEs, widely adopted in Computational Fluid Dynamics (CFD). We briefly summarize the key idea of this method: the domain of the problem is discretized and the solution values are computed by enforcing the conservation law for these small "cells". By converting each volume integrals containing a divergent term to surface integrals (*divergence theorem*), the flux at the surface of each finite volumes is evaluated. Because the flux entering in a given finite volume is equal to that leaving the adjacent finite volume, this method is conservative. Moreover, because it does not need assumption about the mesh structure, the FV method can manage unstructured grid.

3.2.3 OpenFOAM

OpenFOAM [11] is an open-source software, written in C++, that computes the numerical solution of PDEs. It implements several customizable solvers based on FV method, suitable to the most common simulations, and also many applications for the pre-processing and the post-processing. Moreover, it provides a parallel version of the solvers and of the most expensive application, as example the meshing tool. OpenFOAM does not provide a graphical interface, using a third-party software (Paraview) for output visualization. The configuration is completely done by editing the input files.

3.3 Optimization

Recall the optimization formula:

$$\hat{\Omega}_{\mathcal{O}} = \arg \min_{\Omega_{\mathcal{O}} \in \mathcal{O}_{\text{ad}}} \mathcal{J}(\Omega_{\mathcal{O}}, y(\Omega_{\mathcal{O}})). \quad (3.14)$$

The optimization's challenge is find the best shape by evaluating the smaller number of configurations. To achieve this, many algorithms have been designed; in our work, we have used and analyzed two of the most popular algorithms.

3.3.1 Algorithms

Gradient-based methods

The gradient-based methods are iterative methods that, given a function $F(x)$, find a local minimum by taking steps in the direction where function decreases. Starting from an initial guess x_0 , this method evaluates the gradient at this point and it uses this information to compute the next step, repeating this task until the stopping criteria have been satisfied. A generic formulation is:

$$x_{k+1} = x_k + S_k \quad (3.15)$$

where S_k refers to the direction of the movement. In this way, a series of points x_0, \dots, x_n is generated such that $F(x_0) \geq \dots \geq F(x_n)$. The movement S_k determines the converging velocity.

Method	S_k
Gradient descent method	$-\gamma \nabla F(x_k)$
Newton method	$-\gamma \mathbf{H}^{-1} F(x_k)$

Above, we reports two examples of gradient-based method, where γ refers to the step size. The gradient descent method executes the steps by moving in the opposite direction to the gradient at each point x_k , while the Newton method uses also the Hessian matrix to achieve a more precise movement towards the local minimum. For this, the Newton method requires a smaller number of iteration to converge, but it also implies an additional cost to build the Hessian matrix.

The gradient-based methods are highly efficient for the local minimum searching, but totally useless in a global optimization problem. Moreover the gradient-based methods need the evaluation of the function's gradient. When an analytical gradient is impossible to reach, as in our work, the numerical approximation is mandatory. Using an approximation, a small error is always introduced and the method lose accuracy.

Genetic algorithm

Inspired by the process of natural selection, the genetic algorithm is commonly used for optimization problem. The main idea of this method is that a population of a candidate solutions, called individuals, evolves toward better solutions, by mutating the own properties.

The evolutionary process starts from a population composed by random individuals and the fitness of each of them is evaluated. The fitness is usually the objective function used in the optimization problem. The more fit individuals are selected and they are recombined and randomly mutated to compose a new population. This selection process is computed iteratively, and at each iteration (*generation*) a new population based on the best individuals is created. At the end, after a suitable number of generations, the population contains the optimal individuals.

The genetic algorithm requires a lot of iterations to converge to the minimum, but it allows to find the global minimum without assumption on the function to optimize.

3.3.2 Dakota

Dakota [12] is a toolkit for system design and analysis. It provides an extensible and flexible interface between the computational methods and the iterative analysis methods; its purpose is to make easy the analysis or the optimization of a simulation code. Dakota implements algorithms for parameter study, design of experiments, optimization (gradient-based, derivatives-free, global optimization). Written in C++, it also supports parallel computations.

To be suitable for several fields of study, Dakota acts as *black box*: for each simulation, it creates a new directory, changing properly the parameter values, and running the user code.

Chapter 4

Reduced Order Method Injection

After a detailed discussion about all the components of the original optimization system, we present in this chapter how this has been modified by adding the Reduced Order Method (ROM) in order to make the optimization computationally efficient.

It is possible to approximate the Partial Differential Equation (PDE) solution for a new given parameter value by combination of suitable solutions already computed. This idea can be exploited in our optimization problem: avoiding an (expensive) high-fidelity solution for each iterations of the minimization procedure, a reduced basis space is generated from some numerical solutions and the object function is evaluated on the reduced basis solutions. We remark the optimization formulation

$$\Omega_0 = \arg \min_{\Omega \in \mathcal{U}_{ad}} \mathcal{J}(\Omega, y^{\mathcal{N}}(\Omega)), \quad y^{\mathcal{N}}(\Omega) \in \mathbb{X}^{\mathcal{N}} \quad (4.1)$$

where $\mathbb{X}^{\mathcal{N}}$ denotes the discretized space; insertion of the reduced basis approximation in (4.1) yields

$$(\Omega_0)_N = \arg \min_{\Omega \in \mathcal{U}_{ad}} \mathcal{J}(\Omega, y_N^{\mathcal{N}}(\Omega)), \quad y_N^{\mathcal{N}}(\Omega) \in \mathbb{X}_N^{\mathcal{N}}. \quad (4.2)$$

Hence, the optimization procedure needs to compute many Online phases, much faster than the high-fidelity solver. Several PDEs have to be solved numerically just to create the reduced basis space. The ratio between the number of iterations required by the optimization procedure (typically high) and the number of solutions used for space generation (typically small) gives us the ideal speedup using ROM, assuming that the solver is the most relevant component from the computational point of view.

4.1 EZyRB

EZyRB [10] is a library for the Model Order Reduction based on Proper Orthogonal Decomposition (POD) for the reduced space generation. It is written in Python and it aims to provide an easy but generic interface; moreover it is completely independent from the solver used, working as a *black box*:

indeed library can handle different file types, extracting the chosen "output" and using it for space generation.

EZyRB provides all the ROM features: starting from several files, it creates (and stores) the reduced basis space; then, the final user can query for a new parametric point and the solution approximate is returned. Moreover, a customizable error estimation has been implemented (see Section 2.4). Also, using this estimator, the library gives the opportunity to interrogate the reduced basis space to find the new parametric point where an high-fidelity solution should be computed. In order to achieve this, EZyRB splits the parametric domain in many simplex and selects the one which has the maximum error. This simplex indicates the region where the reduced basis space is not rich enough: the barycentric center of the simplex is the optimal parametric point where a new solution has to be computed to improve the overall accuracy of the reduced model.

4.2 Mapped Solution

We remark the necessity to deform the original ship surface and to create a new mesh, to maintain an high quality of space discretization (Section 3.1.3). This involves a different number of cells and points in the meshes. We recall the formula for reduced basis space (2.3): the linearly combined solution must have the same dimension. It is obvious that, using different order meshes, the numerical solutions will have a different dimension.

To overcome the problem, we need to rebuild the solutions on a fixed number of cells (or points): a valid strategy to allow this is to project the solutions onto the points defining the original surface.

4.2.1 Mapping algorithm

Project the solution on a new set of element means assigning a value to the new elements by interpolating the values of their nearest neighbor. These elements can be the points or the cells of the mesh, and the common solvers provide a output vector containing the values for each elements. Because the implementation is slightly different using points or cells, we introduce two different algorithm to manage the two possibility.

Point Data Solution

We suppose the vector $y^{\mathcal{N}}(\mu)$ refers to the numerical solution expressed for each point of the mesh, where \mathcal{N} is the number of points. We want to map this solution on the \mathcal{M} points defining the original surface; $P_{\mathcal{N}}$ and $P_{\mathcal{M}}$ denote respectively the set of solution points and the set of surface points. The main idea of this implementation is assign a new value to point $p_i \in P_{\mathcal{M}}$ looking for the its K neighbors in $P_{\mathcal{N}}$; after selecting the values of the neighbors from the solution vector, the result of the interpolation of neighbors values is assigned to the point p_i . The procedure is repeated for all the surface points. The technique is summarized in Algorithm 4.1: here the function FINDKNEARESTNEIGHBORS, EXTRACTDATA and INTERPOLATE are

Procedure 4.1 Algorithm to map the solution onto the geometry points

Input:

the vector $y^{\mathcal{N}}$ of the points coefficient of the solution
the set $P_{\mathcal{N}}$ of solution points
the set $P_{\mathcal{M}}$ of surface points
the number k of neighbors

Output:

the vector $y^{\mathcal{M}}$ of the points coefficient of the mapped solution

```

1: function MAPSOLUTIONBYPOINT( $y^{\mathcal{N}}, P_{\mathcal{N}}, P_{\mathcal{M}}, k$ )
2:    $y^{\mathcal{M}} \leftarrow \{\}$ 
3:   for each  $p_i \in P_{\mathcal{M}}$  do
4:     neighbors  $\leftarrow$  FINDKNEARESTNEIGHBORS( $p_i, P_{\mathcal{N}}, k$ )
5:      $V_{neigh} \leftarrow \{\}$ 
6:     for each neighbor  $\in$  neighbors do
7:        $V_{neigh} \leftarrow V_{neigh} \cup \text{EXTRACTDATA}(P^{\mathcal{N}}, \text{neighbor})$ 
8:     end for
9:      $V_{new} \leftarrow \text{INTERPOLATE}(V_{neigh})$ 
10:     $y^{\mathcal{M}} \leftarrow y^{\mathcal{M}} \cup V_{new}$ 
11:  end for
12: end function

```

not specified because they are dependent from the data structure used in the implementation. We limit our-self to roughly explaining our idea:

FINDKNEARESTNEIGHBORS: It is an expensive *search by value* operation; this function has to compute the distance between the point p_i and all the points in the solution. Moreover, the function has to be called for each point in the surface. In order to compute this operation as fast as possible, we use a *k-d tree*. The space of the solution points is splitted recursively in many subcells, and a tree is built according to three principles: (i) each internal node of the tree corresponds to a cell, (ii) each leaf corresponds to a point and (iii) if a cell contains a subcell (or a point), the corresponding node has to be the parent of the subcell nodes (or of the point leaves). Because of the hierarchical structure, the neighbors search becomes really fast. Moreover, considering that the points do change coordinates during procedure, the tree has built just one time.

EXTRACTDATA: It is an easy *search by index* operation; supposing the indices, or the coordinates, of the neighbors is already known, the function just selects the correct element in the solution vector.

INTERPOLATE: It provides a new value from the interpolation of the neighbors values. It is not strictly dependent from the data structure used, but from the user choice. In our case, we use the Inverse Distance Weighting:

$$V^{new} = \begin{cases} \frac{\sum_{i=1}^k w_i V_i^{neigh}}{\sum_{k=1}^k w_i} & \text{if } d(V^{new}, V_i^{neigh}) \neq 0, i = 1, \dots, k \\ V_i^{neigh} & \text{if } d(V^{new}, V_i^{neigh}) = 0, i = 1, \dots, k \end{cases} \quad (4.3)$$

where the weights $\{w_i\}_{i=1}^k$ are defined

$$w_i = \frac{1}{d(V^{new}, V_i^{neigh})^p}, \quad i = 1, \dots, k. \quad (4.4)$$

The notation $d(V^{new}, V_i^{neigh})^p$ denotes the distance between the two points, where p is a positive number called power parameter.

Procedure 4.2 Algorithm to map the solution onto the geometry cells

Input:

- the vector $y^{\mathcal{N}}$ of the cells coefficient of the solution
- the set $C_{\mathcal{N}}$ of solution cells
- the set $C_{\mathcal{M}}$ of surface cells
- the number k of neighbors

Output:

- the vector $y^{\mathcal{M}}$ of the cells coefficient of the mapped solution

```

1: function MAPSOLUTIONBYCELLS( $y^{\mathcal{N}}, P_{\mathcal{N}}, P_{\mathcal{M}}, k$ )
2:   centroidssur  $\leftarrow \{\}$ 
3:   centroidssol  $\leftarrow \{\}$ 
4:   for each  $c_i \in C_{\mathcal{M}}$  do
5:     centroidssur  $\leftarrow$  centroidssur  $\cup$  COMPUTECENTROID( $c_i$ )
6:   end for
7:   for each  $c_j \in C_{\mathcal{N}}$  do
8:     centroidssol  $\leftarrow$  centroidssol  $\cup$  COMPUTECENTROID( $c_j$ )
9:   end for
10:  MAPSOLUTIONBYPPOINT( $y^{\mathcal{N}},$  centroidssol, centroidssur,  $k$ )
11: end function

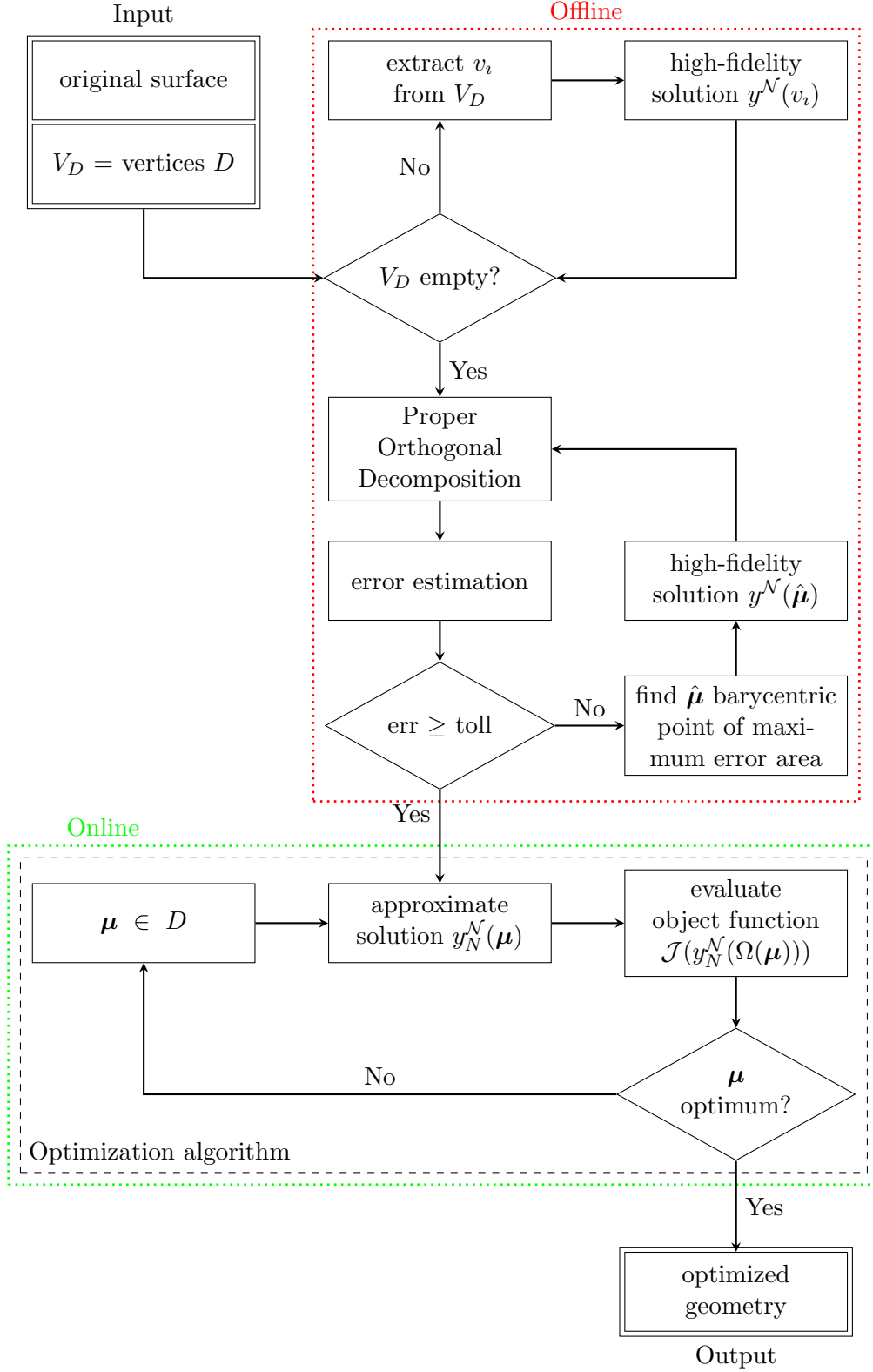
```

Cell Data Solution

Here the coefficients of the solution refer to the output value at each cell of the mesh. In this case the procedure is quite similar to the previous one: the difference consists to an initial phase where the centroid of each cell is computed. In this way, looking for the neighbors of all the centroids becomes easy find the neighbors of the cells (Algorithm 4.2).

4.2.2 Error Handling

The mapping procedure produces inevitably an error: the interpolated solution and the original one have a different number of points and cells, with different coordinates, so differences always occur. Nullify these it is impossible, but customizing the routine we can handle the error to reach the target precision. Basically, the mapping error can be reduced 1) using a more sophisticated interpolating function, 2) using an higher number of neighbors or 3) increasing the number of points and cells. The first two options require just a small change in the above mentioned algorithm, while the increase of points and cells needs additional operations. The idea to achieve this is simple: each cell of the geometry must be splitted into a fixed number of cells (Figure 4.2). We call the ratio between the new number of cells and

Figure 4.1: The *global* algorithm for the reduced space improvement.

the old one intensification factor. The interpolating the solution onto a more refined geometry restricts the loss of information made by the mapping and

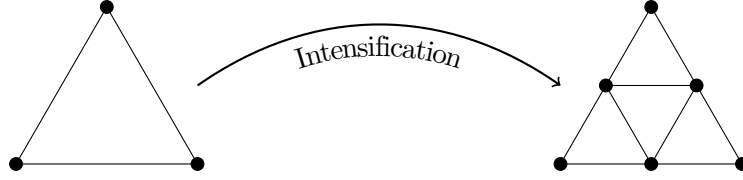


Figure 4.2: Basic step to increase the number of cells and points

maintains at the same time an equal number of cells (or points) for all the parametrized shapes.

The intensification of the number of cells, a larger number of neighbors and a more expensive interpolating function obviously increase the computation time of the mapping procedure, but allow to control the mapping error to make it suitable to all the cases.

4.3 Reduced Basis Space Improvement

The reduced basis methods provide a approximation of the solution and an a posteriori error estimation to manage the accuracy. In this section, we introduce two different strategies to build an accurate reduced space using the minimum number of basis: the first one aims to minimize the global error, while the second increases the precision of the approximation only in a local region of the space.

4.3.1 Global

We briefly remark the a posteriori error estimation strategy: given the set of parametric points where the high-fidelity solutions are computed for the space generation, we can check the difference between these solutions and the reduced basis approximation at those parametric points (see Chapter 2 for more details). The biggest error computed in this way provides us an good estimation for the reduced space accuracy.

The idea of the global improvement exploits the error estimation to determine the region of the reduced basis space where the approximation accuracy is the worst. An high error in the Galerkin Projection means that the space, in that region, has not enough information about the original problem. Hence, by adding a new solution computed in a point belonging to the *maximum error* region, we can enrich the space and reduce the error. Not only: choosing a point in the less accurate region, we ensure an optimal error reduction.

This procedure iterate until the estimate error is less than the given tolerance (flow chart at Figure 4.1): we start from the 2^d vertices of the Free-form Deformation (FFD) domain D and we add each time the high-fidelity solution computed at the barycentric point of the maximum error simplex. It is important to underline that POD is computed at each iteration. Finally, an accurate reduced space is generated and we can approximate the solutions in a optimization cycle to minimize the object function.

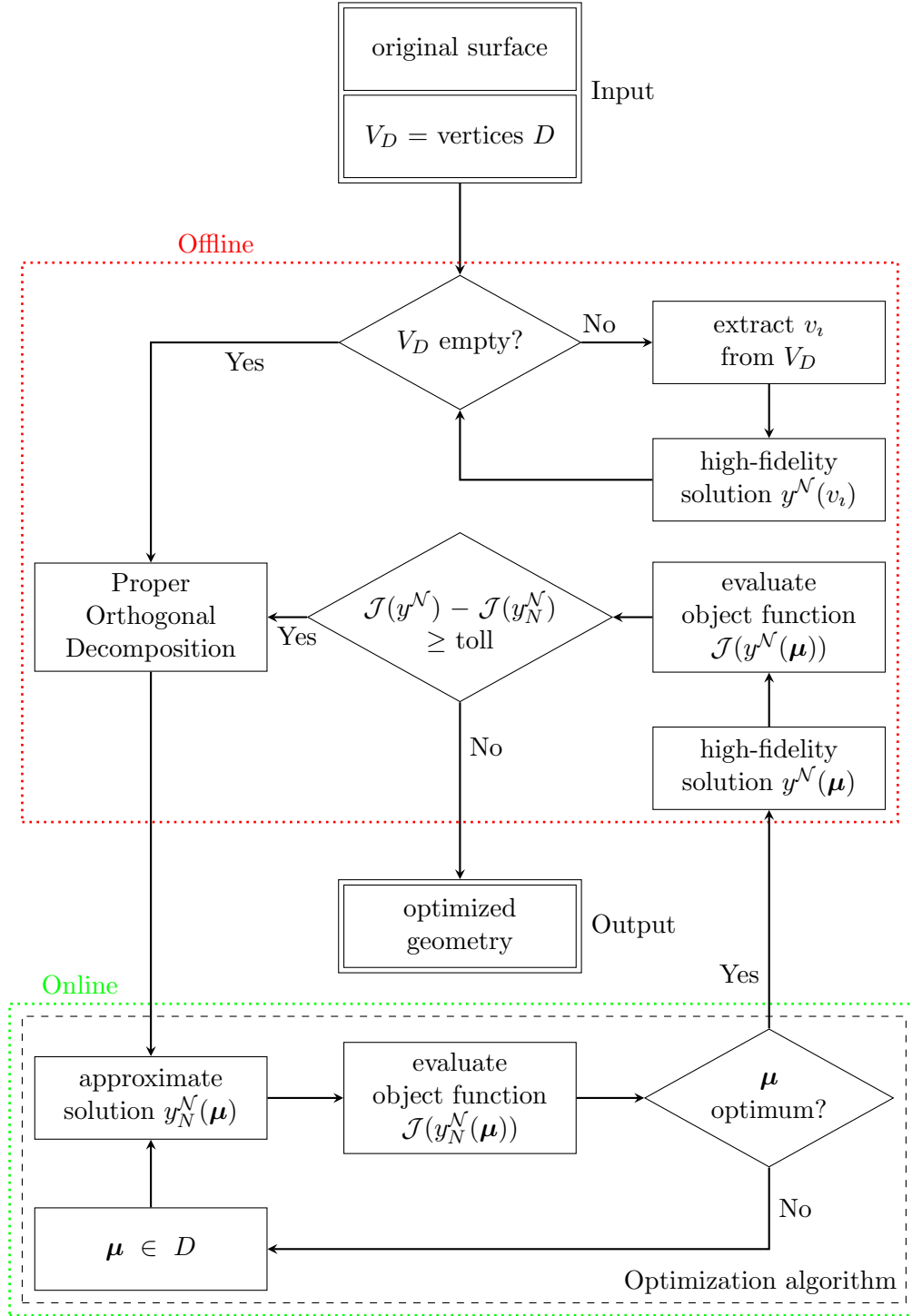
4.3.2 Local

The previous method aims to an optimal reduction of the overall error; in many cases the possibility to reach a good approximation of the solution in all the parametric points is necessary, but in some others this can be useless (and inefficient). For example, we focus on the optimization problem: because we are interested to minimize the objective function, we need a negligible error only for the regions of the reduced space that produce solutions close to the minimum. By adding solutions distant from the area of interest, the accuracy improvement in this area is not relevant.

For this kind of problems, a global improvement is ineffective: several high-fidelity are computed for the error reduction, but many of these do not contribute to increase precision of a limited set of approximated solutions. Hence, a locally enrichment is a nice solution.

First of all, the high-fidelity solutions are computed at the 2^d vertices of the domain D and a reduced basis space is generated. Then, the Galerkin projection acts iteratively on this space in order to obtain the optimal approximated solution. This solution is validated with an high-fidelity solver and the difference between the objective function evaluated on the reduced solution and on the numerical solution is computed: if it is lower than a given tolerance, the approximated optimum becomes the real optimum. Otherwise, the high-fidelity solution, computed for the validation, is added to the reduced space and the routine restarts.

Figure 4.3 summarizes this method. We underline some features: at each iteration both the Offline and the Online phase are executed; at each iteration, a optimization procedure is completely run; the stopping criterion is based on the difference between the numerical solution and the reduced one, not anymore on an estimated error. Anyway, the best gain of this method is a rapid enrichment of the interested area of reduced space, allowing a correct approximation using even less high-fidelity solutions.

Figure 4.3: The *local* algorithm for the reduced space improvement.

Chapter 5

Results

5.1 Configuration

For the system test, two different architectures have been used:

- A single node, equipped with processor Intel[®] Xeon[®] CPU E5-2650 v2 @ 2.60GHz, 16 cores, where inexpensive tasks (parametrization, optimization) have been performed;
- Two nodes, equipped with processors Intel[®] Xeon[®] CPU E5-2680 v2 @ 2.80GHz, 20 cores per node, where the Computational Fluid Dynamics (CFD) simulations (including the meshing task) have been performed.

Regarding the software, we use OpenMPI 1.8.2 as message passing library and the Atlas library for the linear algebra.

5.2 Parametrization

The big challenge in the deformation of the bulbous is the achievement of a parametrization capable to explore a variety of shape without producing inflections in the surface. To reach this target, we apply sequentially two FFD, with a different number of control points and a different domain D .

	Control Points			Number of μ
	x	y	z	
<i>first</i>	5	3	3	3
<i>second</i>	4	4	3	2

Figure 5.1 shows the position of the control points; the first deformation stretches the height and the length of the bulbous, while the second one increases (or decreases) the thickness of the bulbous middle frame. Adopting this strategy, we are able to preserve the surface derivatives continuity and, at the same time, generate many different shapes, just touching few points of these lattices. We use one parameter for the bulbous length, two parameters for the thickness and two for the height (they refer to different sections); we remark that these parameters do not indicate the deformation of the geometrical properties but to the points movement.

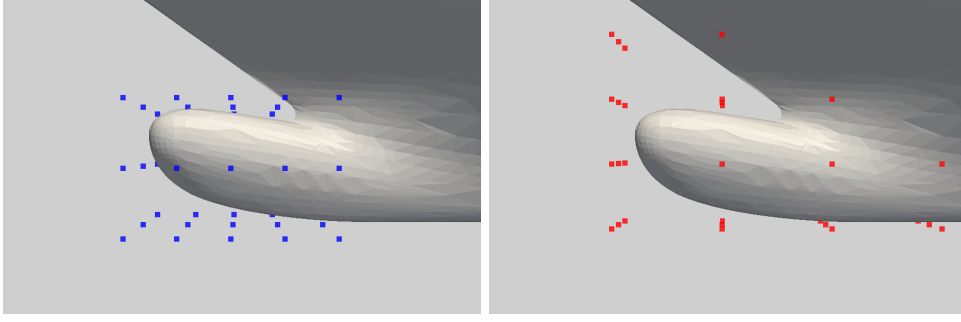


Figure 5.1: Lattices of points used in the sequential deformations

5.3 Mapped solution

In this section, we summarize the result of the solution interpolation on the geometry points (Section 4.2.1), especially taking into account the error introduced from this mapping. We remark it is possible handling the error comes to this interpolation by the configuration of the neighbors values used in interpolation function and the intensifying factor.

Because the high-fidelity solver (OpenFOAM) is cell-centered, we use the cell data to map the solution on the surface. To measure the error, we compare the objective function evaluated on the original solution and on the interpolated one.

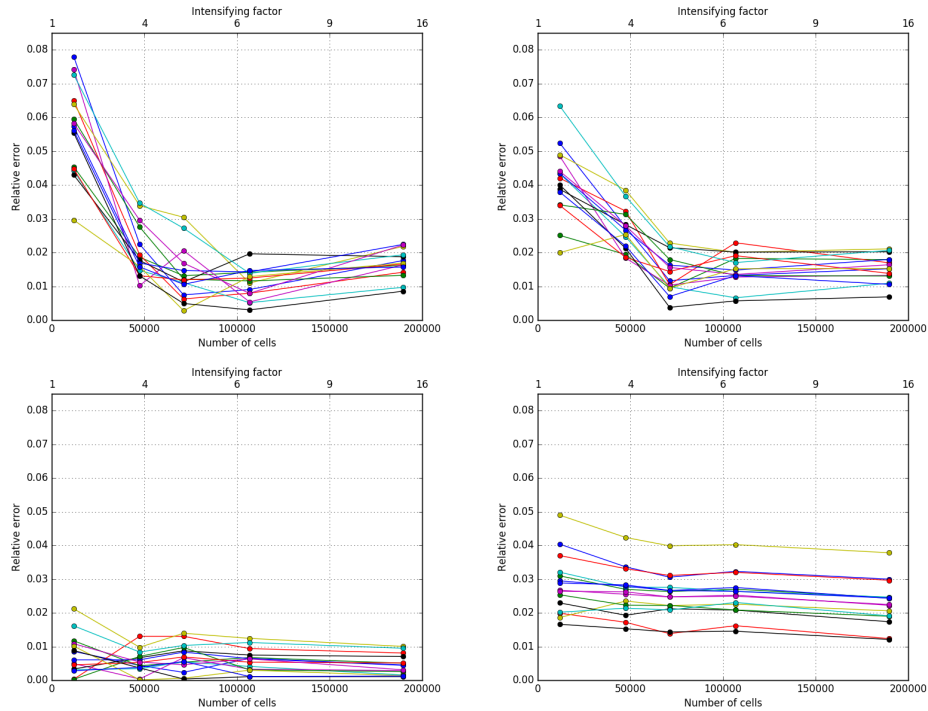


Figure 5.2: Relative error of the objective function evaluated on interpolated solution; (top left) one neighbor, (top right) two neighbors, (bottom left) four neighbors, (bottom right) sixteen neighbors

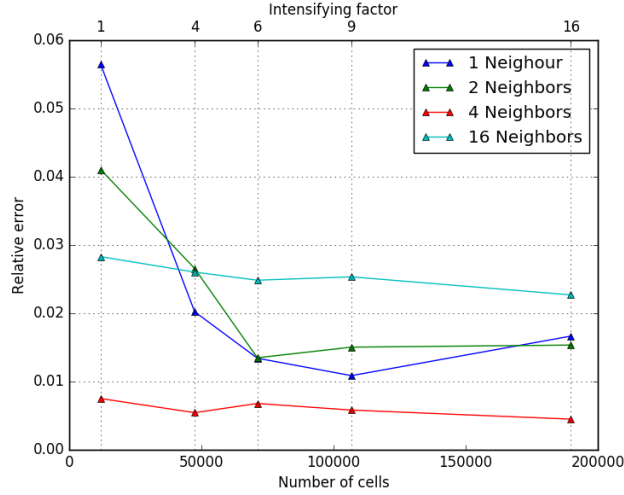


Figure 5.3: Average relative error introduced by mapping procedure, using different neighbors and different number of cells

We run different test case, increasing the number of cells thanks to the intensification procedure, using 1, 2, 4 or 16 neighbor values for the interpolation function. Figure 5.2 shows the results collected for 15 different tests: as expected, the error tends to decrease using more cells and more neighbors. However, the minimum error does not occur using the maximum number of neighbors: all the tests computed using 4 neighbors show the best behavior. The increase of the number of cells improves the accuracy of the mapping technique, but it is important to note that by using few neighbors (1-2), all the tests reach the minimum error with approximately 75000 cells.

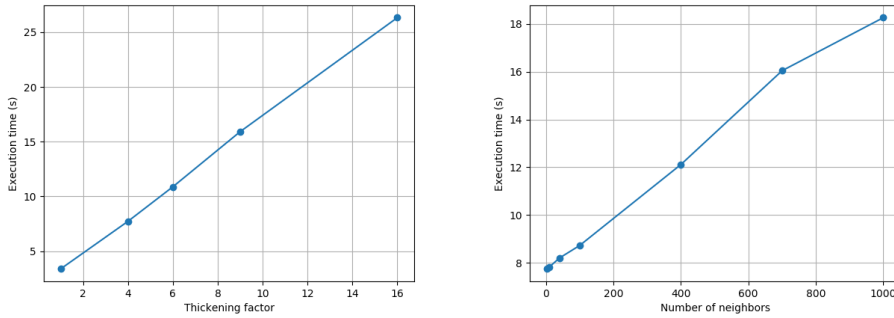


Figure 5.4: Execution time of mapping procedure using different number of cells (left) and different number of neighbors (right)

In Figure 5.4 we report the execution time of the mapping procedure; with a fixed number of neighbors, we measure the time for the complete routine increasing the intensifying factor. Then, with a fixed number of cells, we measure the time changing the number of neighbors. The scaling is linear in both tests, thanks to the fast nearest neighbors search permitted by the tree data structure; however, it is important to note that the execution time increases slower with the growth of the number of neighbors than with the

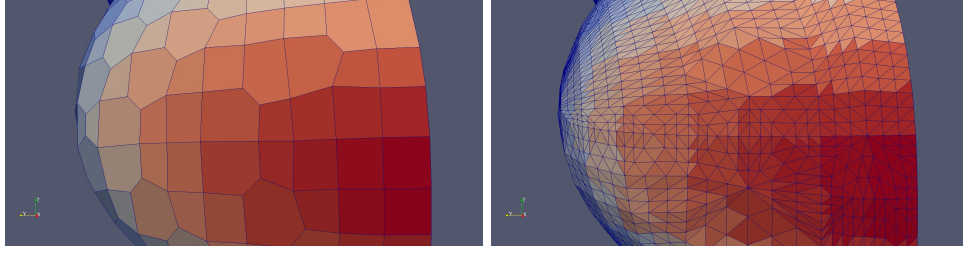


Figure 5.5: Screenshots of the original solution (left) and the interpolated one (right)

enhancement of number of cells.

5.4 Solver

We recap below the main setting of the CFD simulation. We remark the need of build the mesh from scratch to each different geometry, thus all the simulations are preceded by the meshing phase. The tool used to build the mesh is **snappyHexMesh**, provided by OpenFOAM toolkit.

application	snappyHexMesh
application	interFoam
model	kOmegaSST
delta time	0.001s (adjustable)
max delta time	0.025s
simulation time	60s

We recursively use the mesh tool in order to refine the region of the discretized space around the ship, creating a thick boundary layer near the surface. After this procedure, the number of cells composing the mesh is approximately 8 millions. The execution time of this kind of simulation, comprehensive of the mesh construction, is more or less 7 hours.

5.4.1 Reduced Basis Optimization

The two different strategies for the accuracy improvement of the reduced basis space have been tested. First, we adopt the *global* idea, estimating the error at each iteration. We can see (Figure 5.6) some fluctuations in the error trend, but anyway it tends to decrease. However, it is important to note that adding 18 reduced basis the estimate error gets an improvement of only $\approx 0.3\%$. Also because of the number of parameters (5), the global error's decrease is really slow; to obtain a precise approximation on all the parametric domain, many solutions has to been involved, making partially useless the ROM injection.

Starting from the reduced space improved by the global strategy, we still improve the accuracy of the approximation by the *local* technique. We briefly recall the idea behind: we gain more precision by adding to the space the high-fidelity solution computed on the optimal parametric point found by

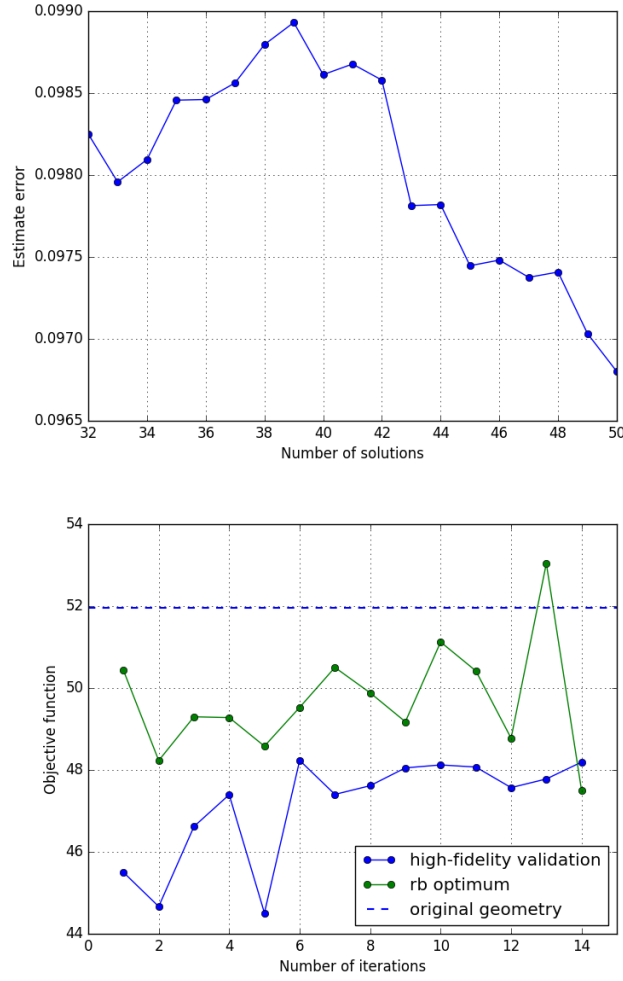


Figure 5.6: Top: the reduced estimated error using the global technique with an increasing number of high-fidelity solution involved; Bottom: the difference between the objective function computed on the reduced solution and the high-fidelity solution in the local improvement method.

an optimization routine. So, at each iteration we have an approximated optimum design and the high-fidelity validation: we present in Figure 5.6 the values of the objective function evaluated on the reduced basis solution, and on the numerical one. The difference between these values is fluctuating, but we can note an appreciable improvement of the velocity with which the error decrease. As reported in Table 5.1, the error initially is 9.77%, aligned with the estimation done in the global improvement. After 14 iterations, the stopping criteria (relative error less than 1.5%) has been reached and the relative error is considerably reduced. Of course, the measure does not reflect the global accuracy of the space, but because the optimal design oscillates between a limited region of the space, we can achieve a fast improvement by enriching this region. In fact, the procedure produces a shape with a better resistance of $\approx 8.5\%$ compared to the original.

As regards the optimization used in the above mentioned procedure, we

$\mathcal{J}(y_N^N(\hat{\Omega}))$	$\mathcal{J}(y^N(\hat{\Omega}))$	relative error (%)	improvement (%)
45.500	50.43	-9.77	-2.92
44.666	48.23	-7.38	-7.16
46.619	49.298	-5.43	-5.10
47.396	49.279	-3.82	-5.14
44.509	48.581	-8.38	-6.48
48.234	49.528	-2.61	-4.66
47.403	50.503	-6.13	-2.78
47.616	49.883	-4.54	-3.97
48.051	49.177	-2.28	-5.33
48.123	51.124	-5.87	-1.58
48.071	50.419	-4.65	-2.94
47.569	48.776	-2.47	-6.10
47.780	53.043	-9.92	2.10
48.192	47.498	1.46	-8.56

Table 5.1: Results of the local improvement strategy: the objective function on the reduced solutions, the objective function on the high-fidelity solutions, the relative error of the reduced approximation and the percentage of the resistance reduction respect to the initial shape.

tested two algorithms, by applying at the same reduced space: the Newton gradient-based algorithm and the genetic one. The number of iterations (Table 5.2) is greatly different: the Newton method computes 43 iteration to reach the minimum, while the genetic algorithm needs about 2000 iterations to find the fittest design (Figure 5.7). However, the gradient-based algorithm is able to converge to the local minimum: the optimal shapes obtained from these methods have remarkably different parameters. Evaluating the objective function on the two deformed ship, a different resistance is also notable, in favour of the genetic algorithm.

Because of the ROM, also using the evolutionary strategy the execution time of the optimization cycle remains negligible compared to the necessary time of a single CFD simulation. Moreover, the two methods work very well

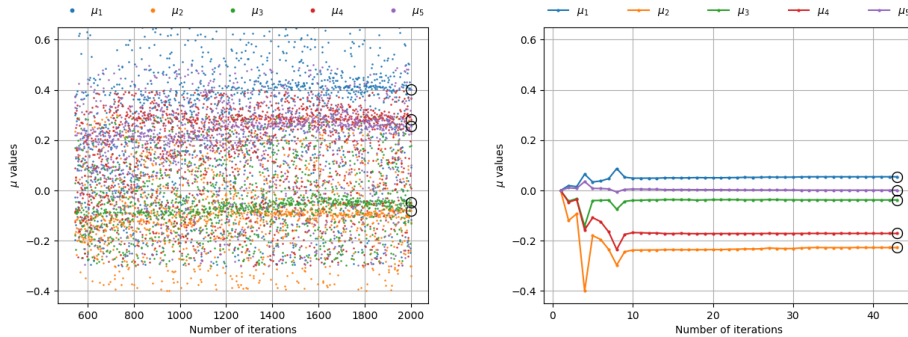


Figure 5.7: Comparison between the genetic algorithm (left) and the Newton method (right).

number of iterations	parameters					$\mathcal{J}(y_N^N)$
	μ_1	μ_2	μ_3	μ_4	μ_5	
43	0.054	-0.227	-0.037	-0.171	0.001	48.774
2003	0.401	-0.080	-0.046	0.283	0.256	48.187

Table 5.2: The comparison between the optimal shape obtained with Newton method (top) and the one obtained with genetic algorithm (bottom).

together: considering the genetic algorithm let the initial population evolves towards the fittest shape, the parametric point returned from the latter can be used as initial guess for the gradient-based algorithm, in order to rapidly converge to the minimum. Figure 5.8 shows the the sequential application of the algorithms; first the evolutionary algorithm explores the parametric space, by identifying the set of parameter values which originate the fittest shape. Then, the Newton method ensures to converge to the minimum.

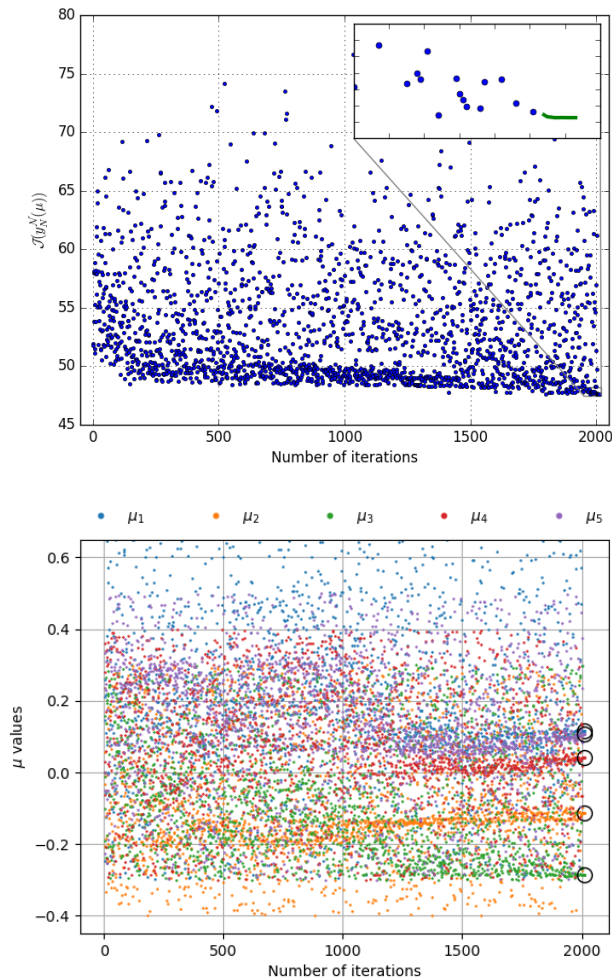


Figure 5.8: Hybrid optimization algorithm: on the top the objective function, with the zoom of the Newton method; on the bottom, the parameter values.

5.4.2 Reduced Basis Error

In this section, we focus on the difference between the reduced basis solution and the high-fidelity one. Thanks to the improvement of the reduced basis space, the two solutions are quite similar (Figure 5.9, 5.10); the bigger errors are in the bulbous peak.

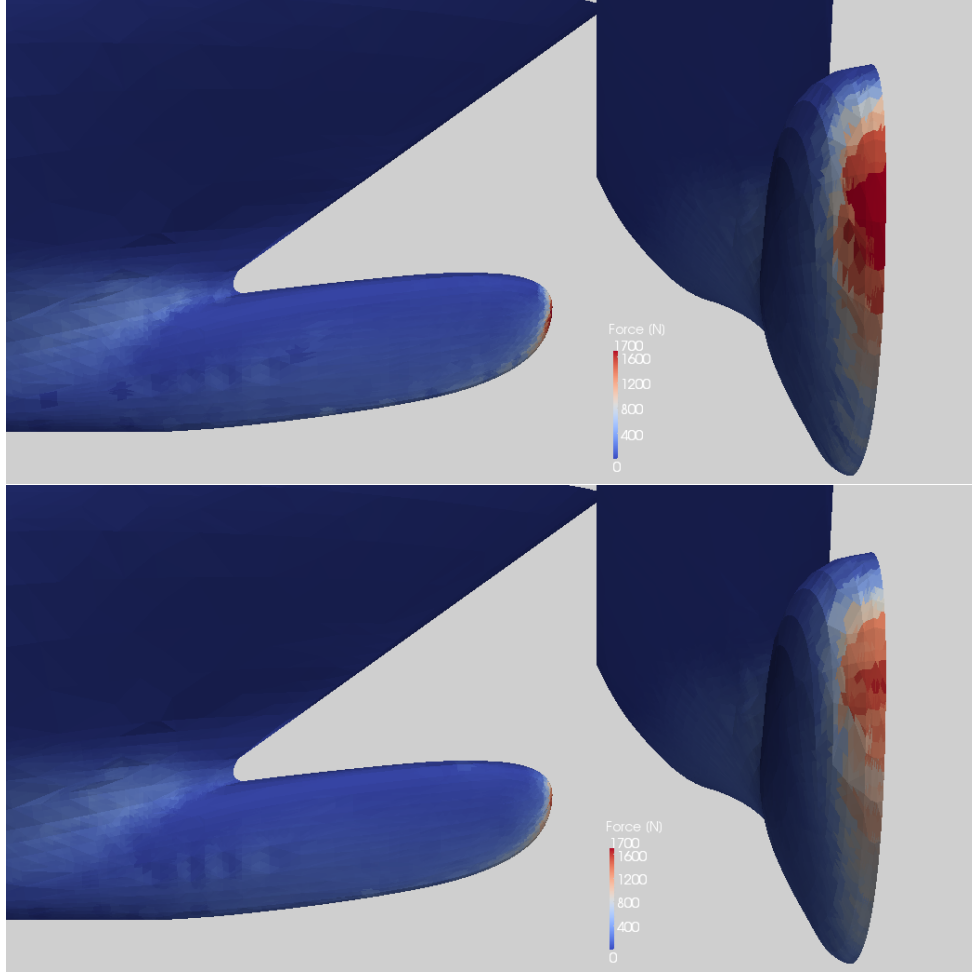


Figure 5.9: Screenshots of the high-fidelity solution (top) and the reduced solution (bottom).

To be more precise, we compute the difference between the two solutions $e^{\mathcal{N}} = y_N^{\mathcal{N}} - y^{\mathcal{N}}$; the vector $e^{\mathcal{N}}$ contains the absolute error of the reduced approximation expressed for each cell. In order to get a precise quantification, we calculate the relative error as the ratio between the error norm and the high-fidelity solution norm. Using different norms, the results are:

	$\ \cdot\ _1$	$\ \cdot\ _2$	$\ \cdot\ _\infty$
<i>relative error</i> $\ y_N^{\mathcal{N}} - y^{\mathcal{N}}\ /\ y^{\mathcal{N}}\ $.0812	.1979	.4050

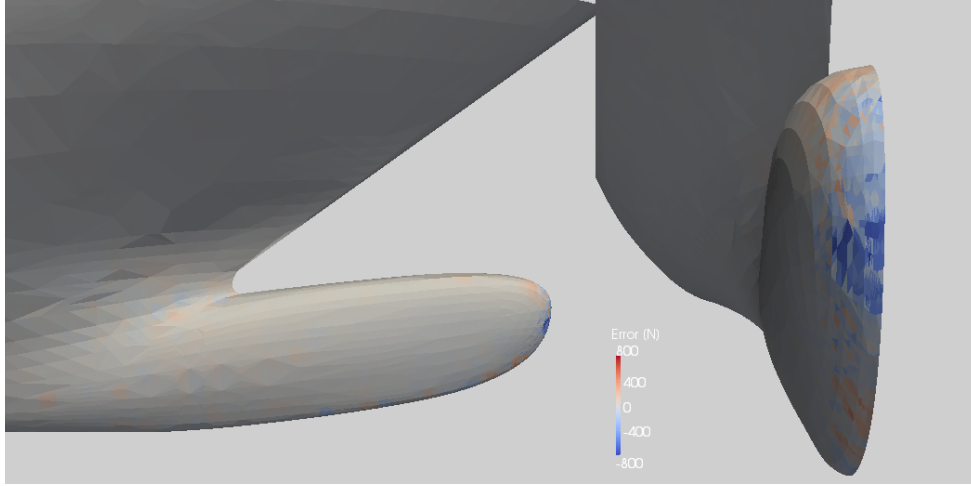


Figure 5.10: Absolute error between the reduced solution and the high-fidelity one.

5.4.3 Reduced Basis Speedup

After validating of the reduced model correctness, we finally try to estimate the concrete speedup by ROM. To have an effective quantification of the reduced time, we should have executed the optimization procedure with and without the reduced basis method, comparing the times and the results. Unfortunately, with no reduced basis method, each iteration of the optimization procedure requires approximately 7 hours; the number of iterations is not known a priori, but we can estimate it with the number of the iterations needed by the ROM system. In our case, the genetic algorithm requires 2000 iterations, while the Newton method has to evaluate the objective function at least 100 times for converge. This implies an estimated overall time equal to 14700 hours, or rather 600 days of computation time. Such a long time makes unusable the system. By involving the reduced basis method, each iteration is composed by the shape deformation, by the Galerkin projection and by the objective function evaluation; because of this, the execution time of a single iteration becomes negligible (≈ 30 seconds) and the complete optimization procedure does not require more than one hour, also thanks to the parallel support of Dakota. The most expensive part is the space generation: because we used a mixed strategy to create and improve the reduced space (global and local improvement), 60 high-fidelity solutions and 14 optimization cycles was been executed. Hence, the overall time is about 434 hours, or rather 18 days.

Chapter 6

Conclusion

In this thesis, we improved a naval shape optimization system by the exploitation of reduced model, in order to achieve a massive computational time reduction.

Primarily, we explored all the open source software used, and we have integrated them to build a automatic optimization system. Then, the reduced basis method was applied to get better performance.

We presented an interpolation procedure to give the possibility of handling meshes with a different degrees of freedom, avoiding the loss of quality due from mesh deformation.

We also presented two different strategies for the increase of the reduced model accuracy: the global algorithm decreases the estimated error in all the space, while the local one aims to improve only a specific region. The main objective of these procedures is to gain better accuracy in the reduced solution using the smallest number of basis.

The test on the reduced system had positive outcomes: the optimal shape of the ship was found, reducing the hull resistance by 8.5%. The reduced basis method made possible an remarkable computational gain: to converging to the best shape, the final system needs less than 450 hours, approximately 38 times faster compared to initial system. Despite the time reduction, the reduced model provides an accurate approximation of the solution: the error between the objective function evaluated on the approximated solution and on the high-fidelity one is about 1.5%.

The reduced order model has proven effective in a naval optimization system. Because of the positive outcomes, the work done for this thesis can continue with the following future developments:

- the hull parametrization can be extended to an other parts of the ship - e.g, the stern hull, the propellers;
- because of the local improvement strategy, the online phase of the reduced basis methods becomes even more relevant from the computational point of view: a parallel implementation of this can yield a considerable gain.

Bibliography

- [1] J. S. Hesthaven, G. Rozza, and B. Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. Springer Briefs in Mathematics, 2015.
- [2] A. Quarteroni and G. Rozza. *Reduced Order Methods for Modeling and Computational Reduction, volume 9*. Springer Milano, MS&A Series, 2014.
- [3] Rozza G, Huynh DBP, Patera AT. *Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations*. Arch Comput Methods Eng. 2008;15:1–47.
- [4] Ballarin F, Manzoni A, Rozza G, Salsa S. *Shape optimization by free-form deformation: existence results and numerical solution for Stokes flows*. J Sci Comput. 2014;60(3):537–63.
- [5] Salmoiraghi F. *Reduced order models for potential flows past parametrized NACA airfoils based on an isogeometric boundary element method*. Master thesis, Politecnico di Milano. 2014.
- [6] Sederberg TW, Parry SR. *Free-form deformation of solid geometric models*. ACM. 1986;20(4):151–60.
- [7] F. Salmoiraghi, F. Ballarin, L. Heltai and G. Rozza. *Isogeometric analysis-based reduced order modelling for incompressible linear viscous flows in parametrized shapes*. Advanced Modeling and Simulation in Engineering Sciences 2016;10.1186/s40323-016-0076-6
- [8] F. R. Menter *Two-equation eddy-viscosity turbulence models for engineering applications* AIAA Journal, 1994; 10.2514/3.12149
- [9] <https://github.com/mathLab/PyGeM>
- [10] <https://github.com/mathLab/EZyRB>
- [11] <https://www.openfoam.com>
- [12] <https://dakota.sandia.gov>